Split Second Motion Blur

Matt Ritchie Greg Modern Kenny Mitchell
Disney Interactive Studios

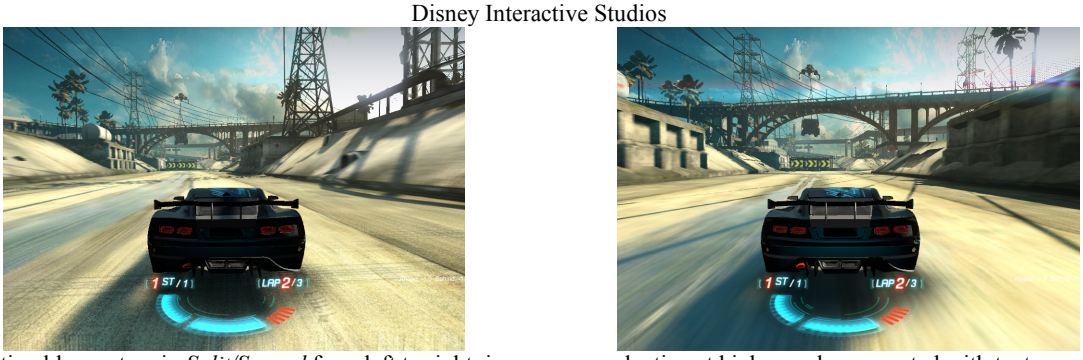

Figure 1: Motion blur system in *Split/Second* from left to right; image space adaptive at high speed, augmented with texture space blur.

Introduction

Motion blur is key to delivering a sense of speed in interactive video game rendering. Further, simulating accurate camera optical exposure properties and reduction of temporal aliasing brings us closer to high quality real-time rendering productions. We describe a motion blur system that integrates image and texture space motion blur for smooth results with less than one sample per pixel. We apply the algorithm in the context of a deferred shading rendering engine used in *Split/Second* (Disney: Black Rock), but the method also applies to forward rendering, ray tracing or REYES style architectures.

Method

Image Space Motion Blur Motion blur may be performed as a post process in image space using scene velocities stored per-pixel [Rosada 2007]. Such vectors are typically held in a packed G-Buffer format as 2D 8-bit components or further packed to an 8-bit vector palette. The averaging filter kernel then performs sheared sampling along the direction of such vectors from a stored rendered image.

The quality of this method depends on the number of samples contributing to the averaging calculation of each resulting blurred pixel. To guarantee artifact free results we require to recover enough samples to satisfy spatial and temporal frequency bounds [Egan et al 2009]. For example, a simple pixel with velocity of 16 pixels per second requires a sampling kernel with 16 samples. With a single image to sample from and no layered depth information, we do not account for temporal occlusion effects.

The speed of this method depends on the bandwidth of generating and storing motion vectors, then recovering image space velocities for blur sampling in the post process. Here, we describe an improved algorithm through the use of vector motion IDs.

Vector Motion IDs For each visible pixel of each rendered object a pixel velocity is stored. Under perspective project image space motion vectors vary greatly across the screen. However, in scenes of high velocity coherence, where world space velocity is constant for the majority of the scene, these motion vectors may be reduced to a shared ID representing the world space velocity. In the post process stage, a recovery of the image space velocity is achieved by inverse projection of the world space velocity matching the sampled motion ID. With 4 bits sufficient to store motion IDs of 16 scene elements with different velocities, this minimizes the storage space and bandwidth required for velocities and reduces time spent in the G-Buffer generation pass.

Tiled Adaptive Sampling With a few scene velocities under perspective projection we observe a typically smooth and well partitioned variation in the magnitude of image space vectors. Applying a fixed size filter kernel sufficient to account for maximum velocity without artifacts results in too many samples overall to perform the effect in real time. Without scope for variable length sampling loops supported by shader model 3.0 graphics hardware, we therefore apply a tiled classification of velocities to select from a small number of pre-optimized shaders with fixed kernel sizes. This classification may be pre-determined for known camera/scene behaviors, calculated from image space bounds of object based velocities or extracted from results of motion ID generation in either a CPU or GPU compute style processing framework.

Texture Space Motion Blur At this point we still have a limit on the velocities we can represent effectively in real-time with high quality. We eliminate this limitation by combining above with texture space motion blur. Loviscach [2005] shows that existing MIP mapping hardware may be used to apply motion blur to textures. While this method samples anisotropically along the motion vector, we use smaller MIP levels primarily to act as a pre-computation of the wider sampling kernel. Thus allowing us to perform less than one motion blur sample per pixel at run-time without blemishes. For fine grained control, we override regular shader sampling calculations to bias the computed texture level of detail based on coarse object velocity and per pixel distance.

Discussion

Combining image and texture space motion blur methods enable us to smoothly blur the scene's content with high quality. Adaptive sampling provides sufficient blur on geometry motions not handled by texture space blurs. In addition, non-anisotropic texture fetches, which would otherwise only blur the textures appearance without direction, are possible when augmented with image space vector motion blur. A run-time frequency analysis of image content may further reduce required number of samples, e.g. on smoothly varying low frequency textures.

EGAN, K. TSENG, Y.-T., HOLZSCHUCH, N., DURAND, F. AND RAMAMOORTHI, R. 2009. Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur. In *ACM Transactions on Graphics* 28, 3.

ROSADO, G. 2007. Motion Blur as a Post-Processing Effect. In *GPU Gems 3,* H. Nguyen, Ed. 575-582.

LOVISCACH, J. 2005. Motion Blur for Textures by Means of Anisotropic Filtering. *EG Symposium on Rendering,* 105-110.

SIGGRAPH 2010, Los Angeles, California, July 25 – 29, 2010.
ISBN 978-1-4503-0210-4/10/0007